МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Алгоритмы и структуры данных»
Тема: Бинарные деревья

Студент гр. 9303 _____ Ахримов А.М.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2020

Цель работы.

Изучить структуру данных бинарные деревья. Познакомиться с раличными вариантами обхода бинарных деревьев.

Задание.

3в) Для заданного бинарного дерева *b* типа *BT* с произвольным типом элементов:

- напечатать элементы из всех листьев дерева *b*;
- подсчитать число узлов на заданном уровне *n* дерева *b* (корень считать узлом 1-го уровня).

Основные теоретические положения.

Дерево – конечное множество *T*, состоящее из одного или более узлов, таких, что

а) имеется один специально обозначенный узел, называемый *корнем* данного дерева;

б) остальные узлы (исключая корень) содержатся в $m>0$ попарно не пересекающихся множествах T_1, T_2, ..., T_m, каждое из которых, в свою очередь, является деревом. Деревья T_1, T_2, ..., T_m называются *поддеревьями* данного дерева.

При программировании и разработке вычислительных алгоритмов удобно использовать именно такое *рекурсивное* определение, поскольку рекурсивность является естественной характеристикой этой структуры данных.

Каждый узел дерева является корнем некоторого поддерева. В том случае, когда множество поддеревьев такого корня пусто, этот узел называется *концевым узлом*, или *листом*. *Уровень* узла определяется рекурсивно следующим образом: 1) корень имеет уровень 1; 2) другие узлы имеют уровень, на единицу больший их уровня в содержащем их поддереве этого корня. Используя для уровня узла *a* дерева *T* обозначение *уровень* (*a*,*T*), можно записать это определение в виде

$$\textit{уровень}\,(a,T) = \begin{cases} 1, & \text{если } a - \text{корень дерева } T \\ \textit{уровень}\,(a,T_i)+1, & \text{если } a - \text{не корень дерева } T \end{cases}$$

где T_i – поддерево

корня дерева T, такое, что $a \in T_i$.

Выполнение работы.

Данная структура бинарного дерева основана на массиве.

Рассмотрим класс binTree. У него есть массив узлов Node, размер массива sizeArray и индекс корня root. Node состоит из поля с данными data и индексами на правое и левое поддерево – RSub и LSub. Методы класса binTree, подразумевающие обход дерева, реализованы рекурсивно.

В методе writeLists осуществляется прямой обход дерева и, если у узла нет поддеревьев(т.е. узел является листом), то его данные выводятся в терминал.

В numberNodes также реализован прямой обход дерева, но до заданного уровня. После просмотра всех узлов с соотвествующим уровнем, метод вывед сумму всез узлов на конкретном уровне.

Входные данные представлены в виде скобочной записи. Дерево можно посмотреть в скобочном виде (метод write) и в виде уступчатого списка (метод view).

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
1.	(A(B(C)(D^(E)))(F(G))) Level = 3	Leafs data: level = 3 leaf data = C level = 4 leaf data = E level = 3 leaf data = G Level = 1 data = A sum = 0 Level = 2 data = B sum = 0 Level = 3 data = C sum = 0 Level = 3 data = D sum = 1 Level = 2 data = F sum = 2 Level = 3 data = G sum = 2 Number of nodes at level 3 = 3

2.	(a(b(d^(h))(e))(c(f(i)(j))(g^(k(l))))) level = 4	Leafs data: level = 4 leaf data = h level = 3 leaf data = e level = 4 leaf data = i level = 4 leaf data = j level = 5 leaf data = l Level = 1 data = a sum = 0 Level = 2 data = b sum = 0 Level = 3 data = d sum = 0 Level = 4 data = h sum = 0 Level = 3 data = e sum = 1 Level = 2 data = c sum = 1 Level = 3 data = f sum = 1 Level = 4 data = i sum = 1 Level = 4 data = j sum = 2 Level = 3 data = g sum = 3 Level = 4 data = k sum = 3 Number of nodes at level 4 = 4

Выводы.

В ходе выполнения данной работы была изучены бинарные деревья. Был разработан класс binTree, основанный на массиве. Были реализованы рекурсивные обходы бинарного дерева для поставленных задач.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <fstream>
#include <string>
#include <iostream>

using namespace std;

template <typename T>
class Node
{
public:
	T data;
	int LSub = 0;
	int RSub = 0;
};
template <typename U>
class BinTree
{
public:
//BinTree() {};
~BinTree() {
	delete[] arr;
}
void resize(int newSize) {
	Node<U>* newArr = new Node<U>[newSize];
	memcpy(newArr, arr, sizeArray * sizeof(Node<U>));
	sizeArray = newSize;
	delete[] arr;
	arr = newArr;
}
void read(ifstream& in) {
	int index = 0;
	char buf;
	in.get(buf);
	if (buf == '(')
		root = readBinTree(in, buf, index);
}
```

```
int readBinTree(ifstream& in,char prev, int& index) {
	if (prev != '(')
		return -1;
	char buf;
	resize(sizeArray + 1);
	int currentIndex = index;
	in >> arr[currentIndex].data;
	in.get(buf);
	if (buf == '(') {
		arr[currentIndex].LSub = readBinTree(in, buf, index +=
1);
		in.get(buf);
		if (buf == '(') {
			arr[currentIndex].RSub = readBinTree(in, buf, index
+= 1);
		}
	}
	else if (buf == '^') {
		in.get(buf);
		if(buf == '('
			arr[currentIndex].RSub = readBinTree(in, buf, index
+= 1);}
	if (buf == ')')
		return currentIndex;
	in.get(buf);
	if (buf != ')'){ cerr << "Wrong tree!\n"; exit(1); }
	return currentIndex;	}
void read() {
	int index = 0;
	char buf;
	cin.get(buf);
	if (buf == '(')
		root = readBinTree(buf, index);
}


int readBinTree(char prev, int& index) {
	if (prev != '(')
		return -1;
```

```
            char buf;
            resize(sizeArray + 1);
            int currentIndex = index;
            cin >> arr[currentIndex].data;
            cin.get(buf);
            if (buf == '(') {
                  arr[currentIndex].LSub = readBinTree(buf, index += 1);
                  cin.get(buf);
                  if (buf == '(')
                        arr[currentIndex].RSub=  readBinTree(buf,  index  +=
1);
      }
            else if (buf == '^') {
                  cin.get(buf);
                  if (buf == '(')
                        arr[currentIndex].RSub=  readBinTree(buf,  index  +=
1);
       }
            if (buf == ')')
                  return currentIndex;
            cin.get(buf);
            if (buf != ')') { cerr << "Wrong tree!\n"; exit(1); }
            return currentIndex;  }
      void write() {
            if (sizeArray == 0) {
                  cout << "()";
                  return;
            }
            writeBinTree(root);
            cout << endl;    }
      void writeBinTree(int index) {
            cout << "(" << arr[index].data;
            if (arr[index].LSub){
                  writeBinTree(arr[index].LSub);
                  if (arr[index].RSub)
                        writeBinTree(arr[index].RSub);
            }
            else if (arr[index].RSub) {
                  cout << "^";
                  writeBinTree(arr[index].RSub);
```

```
            }
            cout << ")";
      }
      void view() {
            view(root, 1);
      }
      void view(const int& index, const int& level) {
            for (int i = 0; i < level; i++)
                  cout << "  ";
            cout << arr[index].data << endl;
            if (arr[index].LSub)
                  view(arr[index].LSub, level + 1);
            if (arr[index].RSub)
                  view(arr[index].RSub, level + 1);
      }
      void writeLists() {
            int index = root;
            writeLists(index, 1);
      }
      void writeLists(int index, int k) {
            if (arr[index].LSub)
                  writeLists(arr[index].LSub, k + 1);
            if (arr[index].RSub)
                  writeLists(arr[index].RSub, k + 1);
            if (arr[index].LSub == 0 && arr[index].RSub == 0)
                  cout << "level = " << k << " leaf data = " <<
arr[index].data << endl;
      }
      void numberNodes(const int& n) {
            int sum = 0;
            numberNodes(n, root, 1, sum);
      cout << "Number of nodes at level " << n << " = " << sum << endl;
      }
      void numberNodes(const int& n, int index, int currentLvl, int& sum)
{
            cout << "Level = " << currentLvl << " data = " <<
arr[index].data << " sum = " << sum << endl;
            if (currentLvl == n) {
                  sum++;
                  return;
```

```
      }
      if (arr[index].LSub)
            numberNodes(n, arr[index].LSub, currentLvl + 1, sum);
      if (arr[index].RSub)
            numberNodes(n, arr[index].RSub, currentLvl + 1, sum);
}
private:
Node<U>* arr;
int sizeArray = 0;
int root;
};
int main() {
BinTree<char> binT;
char buf;
string s;
ifstream f;
cout << "Read from file?[y/n]\n";
cin >> buf;
switch(buf){
case 'y':
      cout << "Enter a file name.\n";
      cin >> s;
      f.open(s);
      if (!f.is_open())
            return 1;
      binT.read(f);
      binT.write();
      f.close();
case 'n':
      binT.read();     }
binT.view();
cout << "Leafs data:\n";
binT.writeLists();
cout << "Enter a level\n";
int n;
cin >> n;
binT.numberNodes(n);
}
```